Package 'RcppProgress'

January 5, 2017

Maintainer Karl Forner <karl.forner@gmail.com>

License GPL (>= 3)

Title An Interruptible Progress Bar with OpenMP Support for C++ in R Packages

Type Package

LazyLoad yes

Author Karl Forner <karl.forner@gmail.com>

Description Allows to display a progress bar in the R console for long running computations taking place in c++ code, and support for interrupting those computations even in multithreaded code, typically using OpenMP.

URL https://github.com/kforner/rcpp_progress

BugReports https://github.com/kforner/rcpp_progress/issues

Version 0.3

Date 2016-12-14

Imports Rcpp (>= 0.9.4)

LinkingTo Rcpp

NeedsCompilation yes

Repository CRAN

Date/Publication 2017-01-05 14:45:35

R topics documented:

RcppProgress-package 2

Index **5**

RcppProgress-package *An interruptible progress bar with OpenMP support for c++ in R packages*

Description

This package allows to display a progress bar in the R console for long running computations taking place in c++ code, and provides support for interrupting those computations even in a multithreaded code.

Details

When implementing CPU intensive computations in C++ in R packages, it is natural to want to monitor the progress of those computations, and to be able to interrupt them, even when using multithreading for example with OpenMP. Another feature is that it can be done so that the code will still work even without OpenMP support. This package offers some facilities to help implementing those features. It it biased towards the use of OpenMP, but it should be compatible when using multithreading in other ways.

quick try: There are two tests functions provided by the package to get a quick overview of what can be done.

These tests are:

RcppProgress:::test_sequential(max, nb, display_progress) - a sequential code, i.e. single threaded

RcppProgress:::test_multithreaded(max, nb, threads, display_progress) - a multithreaded code using OpenMP

Both tests call the very same function that implements a long computation. The **max** parameter controls the number of computations, and **nb** controls the duration of a single computation, that is quadratic in **nb**. The **threads** is as expected the number of threads to use for the computation. The **progress** parameter toggles the display of the progress bar.

On my platform,

```
system.time( RcppProgress:::test_multithreaded(100, 3000, 4) )
```

is a good start.

c++ usage:

There are usually two locations in the c++ code that needs to be modified. The first one is the main loop, typically on the number of jobs or tasks. This loop is a good candidate to be parallelized using OpenMP. I will comment the code corresponding to the tests included with the package.

```
void test_multithreaded_omp(int max, int nb, int threads
                            , bool display_progress) {

#ifdef _OPENMP
    if ( threads > 0 )
        omp_set_num_threads( threads );
```

```
    REprintf(\"Number of threads=%i\n\", omp_get_max_threads());
#endif

    Progress p(max, display_progress); // create the progress monitor
#pragma omp parallel for schedule(dynamic)
    for (int i = 0; i < max; ++i) {
        if ( ! p.is_aborted() ) { // the only way to exit an OpenMP loop
            long_computation(nb);
            p.increment(); // update the progress
        }
    }
}
```

Here we create a Progress object with the number of tasks to perform, then before performing a task we test for abortion (`p.is_aborted()`), because we can not exit an OpenMP loop the usual way, suing a break for example, then when after the computation, we increment the progress monitor.

Then let us look at the computation function (that is completely useless) :

```
double long_computation(int nb) {
    double sum = 0;
    for (int i = 0; i < nb; ++i) {
      if ( Progress::check_abort() ) // check for user abort
        return -1;
      for (int j = 0; j < nb; ++j) {
        sum += Rf_dlnorm(i+j, 0.0, 1.0, 0);
      }
    }
  }
  return sum;
}
```

Here the only interesting line is the `Progress::check_abort()` call. If called from the master thread, it will check for user interruption, and if needed set the abort status code. When called from another thread it will just check the status. So all the art is to decide where to put his call: it should not be called not too often or not frequently enough. As a rule of thumb it should be called roughly evevry second.

Using RcppProgress in your package:

Please note that we provide the **RcppProgressExample** example package along with this package, located in the `examples` directory of the installed package.

Here are the steps to use RcppProgress in a new package:

skeleton create a package skeleton using Rcpp

```
library(Rcpp)
Rcpp.package.skeleton("RcppProgressExample")
```

DESCRIPTION edit the **DESCRIPTION** file and add RcppProgress to the **Depends:** and **LinkingTo:** lines. e.g.

```
Depends: Rcpp (>= 0.9.4), RcppProgress (>= 0.1)
LinkingTo: Rcpp, RcppProgress
```

MakeVars edit **src/MakeVars** and replace its content by
PKG_LIBS = `$(R_HOME)/bin/Rscript -e "Rcpp:::LdFlags()"` $(SHLIB_OPENMP_CXXFLAGS)
`$(R_HOME)/bin/Rscript -e "RcppProgress:::CxxFlags()"` and
PKG_CXXFLAGS +=-Ilibsrc $(SHLIB_OPENMP_CXXFLAGS) `$(R_HOME)/bin/Rscript -e "RcppProgress:::CxxFlags()"`

c++ code Put your code in **src**. You may for instance copy the RcppProgressExample/src/tests.cpp file in **src**, and RcppProgressExample/R/tests.R in **R**, and try to compile the package (`R CMD INSTALL -l test .`) and execute the tests:

```
>library(RcppProgressExample, lib.loc="test")
>RcppProgressExample::test_multithreaded(100, 600, 4)
```

Using RcppProgress with RcppArmadillo: We also provide the **RcppProgressArmadillo** example package along with this package, located in the `examples` directory of the installed package.

The peculiarity is that you have to include the RcppArmadillo.h header before the progress.hpp RcppProgress header, and add the RcppArmadillo in the LinkingTo: field of the package DESCRIPTION file.

Author(s)

Karl Forner

Maintainer: Karl Forner <karl.forner@quartzbio.com>

See Also

OpenMP API specification for parallel programming: http://openmp.org

Rcpp http://r-forge.r-project.org/projects/rcpp

Examples

```
# these are implemented as examples inside RcppProgress
# check the source code

# the underlying test_sequential c++ function has a progress bar and is interruptible
RcppProgress:::test_sequential(100, 300, 4)

# the underlying test_test_multithreaded c++ function is multithreaded
# , has a progress bar and is still interruptible
RcppProgress:::test_multithreaded(nb = 300, threads = 4)
```

Index

∗Topic **package**
RcppProgress-package, 2

RcppProgress *(*RcppProgress-package*)*, 2
RcppProgress-package, 2